Data stream mining of event and complex event streams: a survey of existing and future technologies and applications in big data

Enterprise Big Data Engineering, Analytics, and Management

Martin Atzmueller
University of Kassel, Germany

Samia Oussena
University of West London, UK

Thomas Roth-Berghofer
University of West London, UK

A volume in the Advances in Business Information Systems and Analytics (ABISA) Book Series

BUSINESS SCIENCE Reference
An Imprint of IGI Global

Published in the United States of America by
Business Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA, USA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Names: Atzmueller, Martin, editor. | Oussena, Samia, 1962- editor. | Roth-Berghofer, Thomas R., editor.
Title: Enterprise big data engineering, analytics, and management / Martin Atzmueller, Samia Oussena and Thomas Roth-Berghofer, editors.
Description: Hershey : Business Science Reference, 2016. | Includes bibliographical references and index.
Identifiers: LCCN 2016006898| ISBN 9781522502937 (hardcover) | ISBN 9781522502944 (ebook)
Subjects: LCSH: Big data. | Database management. | Data mining.
Classification: LCC QA76.9.B45 E58 2016 | DDC 005.7--dc23 LC record available at https://lccn.loc.gov/2016006898

This book is published in the IGI Global book series Advances in Business Information Systems and Analytics (ABISA) (ISSN: 2327-3275; eISSN: 2327-3283)

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 3
Data Stream Mining of Event and Complex Event Streams:
A Survey of Existing and Future Technologies and Applications in Big Data

Chris Wrench
University of Reading, UK

Giuseppe Di Fatta
University of Reading, UK

Frederic Stahl
University of Reading, UK

Vidhyalakshmi Karthikeyan
BT, UK

Detlef D. Nauck
BT, UK

ABSTRACT

Complex Event Processing has been a growing field for the last ten years. It has seen the development of a number of methods and tools to aid in the processing of event streams and clouds though it has also been troubled by the lack of a cohesive definition. This paper aims to layout the technologies surrounding CEP and to distinguish it from the closely related field of Event Stream Processing. It also aims to explore the work done to apply Data Mining Techniques to both of these fields. An outline of stream processing technologies is laid out including the Data Stream Mining techniques that have been adapted for CEP.

INTRODUCTION

Event Stream Processing (ESP) and *Complex Event Processing* (CEP) are increasingly wide and valued fields of study in Big Data Analytics. As the Internet of Things becomes more prominent so do events and the need for new and interesting ways of interpreting them. The purpose of this chapter is to clarify the positions of ESP and CEP within the field of Big Data Analytics and outline the range of Data Mining opportunities within ESP and CEP. This is done by identifying the challenges in the field and describing a range complementary and contrasting approaches to overcome them. Though there are numerous papers on the subject, a collection of this specific application was needed.

On this subject there is a useful body of knowledge spread across a wide area rife with different aliases and synonyms and it is difficult to see how the landscape is laid out. Both ESP and CEP evolved

DOI: 10.4018/978-1-5225-0293-7.ch003

out of necessity and independently from multiple problem domains with their own bespoke vocabulary creating a lack of consensus as to the proper title of the field and its components, a phenomenon labelled "Tower of Babel Syndrome" (Cugola & Margara, 2012).

Events and Event Streams are the focus of much of this chapter. An event can be defined in many different ways but at this point it is simplistic to say an event is a thing that *happens*. An Event Stream is an unbounded series of ordered events which, like all Data Streams, is potentially unbounded (Owens, 2007; Yu, Li, Gu, & Hong, 2011). They are a frequent part of our daily lives and, if monitored and processed intuitively, can be an extremely valuable commodity (Eckert, Oriented, Soa, & Eda, 2009). An Event Stream is effectively a specialised Data Stream and as Big Data teaches us, where there is data there is often information and knowledge to be found (Bramer, 2013).

CEP is the means by which meaningful repeated patterns can be discovered amongst a dynamic collection of low level events. Event Stream Processing is the range of technologies used to process the stream and perform Big Data Analytics. It can be argued that ESP is a specialised form of CEP or the two are different approaches to a similar problem, here again is a debate present throughout the literature.

Event Streams are generated and used in many applications. Those generated by the Stock Market are popular subjects for predictive analytics, the transaction history of users on a website and can be used to optimise said website and predict user behaviour, presenting opportunities for profit from advertisement. *Radio Frequency IDentification* (RFID) tags have become cheaper, smaller and common place in high street shops. Sensors positioned around a shop register these tags and the Event Stream can be used to prevent shop lifting (Li, 2010). A further example is that of intrusion detection in which a system administrator employs CEP to identify an intrusion on a network amongst legitimate traffic in the stream (Axelsson, 2000). There are many more examples to be found from the briefest of research into the topic.

Event Stream Processing is a subtopic of Data Stream Mining which has very similar goals but is a far more clearly understood and well defined field. Data Streams present their own unique challenges (i.e. those associated with the **V**elocity, **V**olume and **V**ariety; Ebbers, Abdel-Gayed, Budhi, & Dolot, 2013) which have been the subject of a great deal of research. These same problems apply to Event Streams so it makes sense to first look at the techniques used in Data Stream Mining.

Studying a stream in real-time enables a system or user to react to events in real-time which is of paramount importance for some applications. It also places special requirements on any stream processing technology. The standard database systems used in the majority of Big Data Analytics are not able to meet these requirements. To address this, the database has been adapted or superseded by the Active Database or the Data Stream Management System (DSMS) along with bespoke stream processing query languages and finally CEP systems. Many of these technologies will be looked at later in this chapter. The chapter will then detail several applications of Big Data Analytics and Machine Learning to ESP and CEP.

DATA STREAMS

What Are Data Streams?

Big Data Analytics is a major field of research due to the explosion of data brought about by large corporations and the Internet. Data appears in many different forms and Data Mining applications are developed to match. Initially data was primarily static. It may have been enormous but it was centralised

and non-volatile. Data Streams, however, are quite different. The volume is still large (arguably larger) but the data is generated at such a rate that it takes on new properties which cause issues with traditional Data Mining techniques used for relatively static data (Golab & Özsu, 2010).

Data Streams are useful as they can describe the state of a real world application, action or thing in real-time. Two different types, distinguished by their purpose and source, have emerged: the Measurement stream (where the property or state of an object is monitored) and the Transactional stream (often where the transactions between two objects or users are monitored) (Chaudhry, Shaw, & Abdelguerfi, 2006). The former may be a reading from a machine used in manufacturing to keep actions within a given tolerance or the temperature from a fusion reactor. The latter may be the mouse clicks (known as *Clickstreams*; Bucklin & Sismeiro, 2015) on a webpage allowing a web administrator to track a user through a website and determine user behavioural patterns and identify the more popular areas (Adi, Botzer, Nechushtai, & Sharon, 2006; Hinze, Sachs, & Buchmann, 2009) . A popular application is tracking financial data such as the stock market whose analysis can prove to be quite lucrative. Analysing these Data Streams or simply tracking one stream against the other allows financial professionals to make predictions as to how a stock will rise or fall (Cugola & Margara, 2012). There are many sources of Data Streams and many good reasons to monitor them and apply Big Data Analytics. Transaction streams are synonymous with Event Streams and in this chapter they will be referred to as such.

Data Streams are a specialised type of data set with a few unique properties. Most notably the Data Stream can potentially be infinite in size making it impractical to store. It can be continuous or it can be sparsely populated as data points or events can enter a system at any rate, this may take the form of 'bursty' data. There are four properties of Data Streams that need to be accounted for in Big Data Analytics: **V**olume, **V**elocity, **V**eracity and **V**ariety (Ebbers et al., 2013)(though other literature may include **V**ariability or even **V**erbosity and **V**iscosity; Desouza & Smith, 2014) .

- **Velocity:** Data is generated at such a rate that an algorithm can only pass the stream once. The Stream is potentially infinite so if an algorithm requires more than one pass it will never be current.
- **Volume:** Closely related to velocity, the total volume of a stream is unknown and potentially too large to process completely. Load shedding and sampling techniques have been developed to combat this.
- **Veracity:** The reliability of streaming data is often poor and in need of scrutinising. A strong Data Mining technique must take this into account.
- **Variety:** Data Streams are typically heterogeneous but are often accompanied by one or more other Data Streams of different types. To get a complete picture of the problem domain all streams may need to be accounted. Data may need to be fused on the fly.

These characteristics make special demands upon the algorithms applied to them. Traditional data mining algorithms are designed to expose a trend or concept concealed within the data. This information is hopefully an informative observation about the data that holds for the whole data set. When dealing with data that changes over time the issue of concept drift arises. Concept drift occurs when this observation is only true for a finite period of time before changing or becoming entirely invalid (Widmer & Kubat, 1996). Data Stream Mining algorithms must be able to adapt to any concept drift in the data.

Issues of Real-Time Data Processing

We have mentioned above that Data Streams arrive and need to be processed in real-time. This is a very general statement that requires examining. How one defines real-time can have a substantial impact on the requirements of a system. Applications will have varying requirements as to what kind of delay is tolerable and whether a system must update in real-time.

There are some applications where a quick reaction is paramount, for instance, too large a delay between one of the hundreds of sensors in a modern car triggering an alert and the appropriate response being taken may result in a loss of life. A long delay between when a pattern is recognised as an item being shoplifted and the item being taken out the door renders the system pointless; the theft has already occurred. However, there must be sufficient time allowed to recognise the pattern before raising an alarm. Both of these examples require a quick response but the order of magnitude of time is very different, from nanoseconds to seconds, though both seem instantaneous relative to human reactions.

In terms of a Data Stream raising an alert it may be that the alert is raised as soon as the anomalous data point is produced, but what if the production time of that data point is dissimilar to the processing time due to delays? Fülöp et al. (2012) identify that their algorithm, though in an early and simplified stage, cannot be real-time as measurements are taken every 30 minutes. If this time is arbitrary and can be reduced to a second or below it is still unclear if it is real-time or just approaching real-time. Processing a stream as each point arrives, where each point has no delay between being produced and being read may be unfeasible, especially for large systems. These systems are often referred to as near real-time (Demers et al., 2007; Elmagarmid, 2005).

Various systems have to contend with a delay between the creating and processing of events and there has been some work done to counter this (Mansouri-Samani & Sloman, 1999; Wei et al., 2009). Particularly with ESP which relies on the strict ordering of events to identify causal patterns, delays in the stream can lead to erroneous patterns being detected or interesting patterns being distorted. This can be mitigated by introducing a delay whilst order is checked which further hampers the timeliness of the algorithm, this will be examined later in this chapter.

As well as mitigating out of order data, Stonebraker, Çetintemel, & Zdonik (2005) propose seven additional requirements for real-time Stream Processing. They are listed below with some additional examples from the literature:

- The ability to keep data *moving,* that is to minimize delays by not storing the data, a large number of data stream systems retain only a current history or a model representation of the stream to process.
- Support for queries using a high level (SQL-like) languages such as CQL (Arasu, Babu, & Widom, 2006) enabling a user to query the stream with reference to its relative properties (such as specifying a window).
- To supply "deterministic and repeatable" results which is dependent upon ordering of events by production time rather than processing time using methods such as punctuation to determine when it is safe to process an entry (Tucker, Maier, Sheard, & Fegaras, 2003).
- Have a readily accessible history of the stream or at least a determined base state or signature that can be used as a status-quo from which anomalies can be compared against. This must be tempered against the first requirement.

- Employ a system to ensure availability and mitigate failures such as those detailed by (Hwang et al., 2005).
- Be scalable to handle increased volume by supporting parallel processing.
- Have minimum overhead and real-time response.

The above are presented as rules to consider to "excel at a variety of real-time processing" though all eight features need to be present for an effective, if bespoke, system. Some are business orientated (supporting high level, SQL-like query languages may be good for ease of use but there are other options explored below), but others, such as the ability to keep data moving, are clearly paramount. Later in this chapter we look at the problems with traditional databases that require polling, where the overhead created makes real-time response difficult, and some technologies developed to combat this. Next we outline some approaches used in Data Stream Mining to enable real-time processing.

Real-Time Processing of Data Streams

Mentioned above are some of the problems with data stream mining that make up the four V's. Velocity and Volume can be addressed using techniques such as sampling and load shedding (Maletic & Marcus, 2010). Sampling is a technique familiar to 'static data miners' and statisticians; it entails the creation of a subset of data points that accurately reflect the set as a whole when the set is too large to process in good time. It is a commonly used form of pre-processing in Big Data Analytics that enables a Data Mining Algorithm to produce results in an effective time frame, however it must be used cautiously to avoid misrepresenting the data set and the risk of losing key data points is ever present. Methods used in stationary data mining include linear sampling and basic random sampling methods. In data stream mining the set is assumed to be infinite which creates problems for processing and storing data points, sampling is a useful tool to help address this (Babcock, Babu, Datar, Motwani, & Widom, 2002). Traditional methods have the advantage of knowing the total size of the set and can use this to create a reliable sample. This is not afforded to Data Stream Mining where the size will constantly be increasing. Below are some of the methods of sampling that have been developed to combat this problem.

- **Sliding Window:** Windowing is a method of keeping a snapshot of the stream. There have been many varieties developed, including the landmark window, the damped window and the titled-time window (Hutchison & Mitchell, 2011). Perhaps the most widely used of these is the sliding window. The sliding window (Figure 1) technique is used to keep a current history of the data stream that moves over time. As new data enters from the *right* of the window, old data is excluded from the *left* as the window moves on. In this way the sample is able to adapt to any concept drift in the data (Tsymbal, Pechenizkiy, Cunningham, & Puuronen, 2008; Widmer & Kubat, 1996). The window may progress with every new item or a given unit of time. Ricardo Vilalta, Ma, & Hellerstein (2001) had success with a time window of 20 minutes. Within the window is a snapshot of the stream that can be used for data mining. This does have the disadvantage of losing the history of the stream as data points are forgotten once the window has moved on, but it is computationally efficient.
- **Reservoir Sampling:** Reservoir sampling (Figure 2) has been directly adapted to streams from static techniques where the total number of points are known in advance (Vitter, 1985). Sampling entails the probabilistic insertion of points from the data stream into a *reservoir* of points. Unlike

the Sliding Window technique, the reservoir contains a history of the stream that grows wider as the stream continues. The initial sample of the reservoir is gradually replaced with newer data points to maintain relevancy and the bias towards newer points can be adjusted according to application. Reservoir sampling opens up the sample to batch processing techniques as well as tailored ESP.

- **Hoeffding Bound:** As mentioned above, sampling any data stream may result in an imperfect representation of the set and may lead to the extraction of misleading information, especially in the case of an undetected concept drift. Sampling from a reservoir or a window still carries this risk. The Hoeffding bound can be used to further mitigate this. It states that the true mean of a random variable within a known range will not differ from the estimated mean after *n* independent observations (Hoeffding, 1963). The *n* in this definition refers to the minimum sample size needed to establish a good estimate of the true mean of the sample and this can be used to great effect when sampling a stream. It is also used in other Data Mining algorithms looked at later in this chapter. Johnson, Muthukrishnan, & Rozenbaum (2005) provide further examples of Data Stream Sampling techniques. After sampling has taken place a number of Data Stream Mining Techniques can be applied to the stream.

Figure 1. Sliding window: data enters the window from the right and leaves from the left maintaining an up–to-date sample within the window. Data that leaves from the left is forgotten.

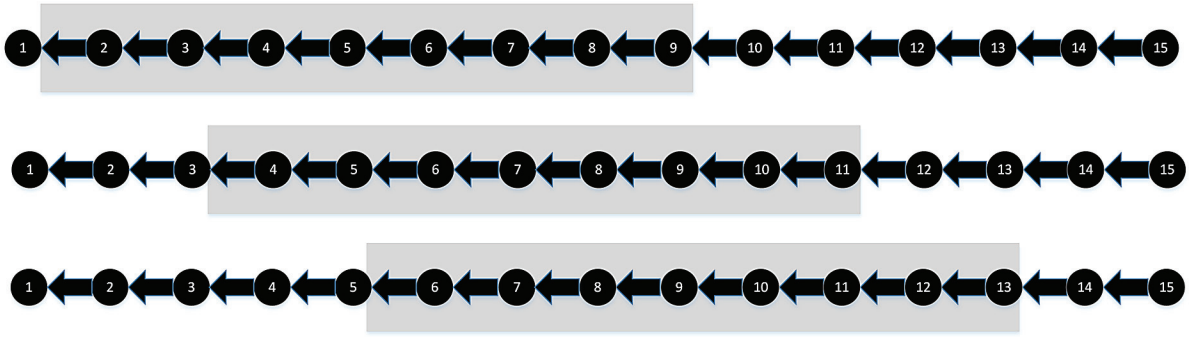
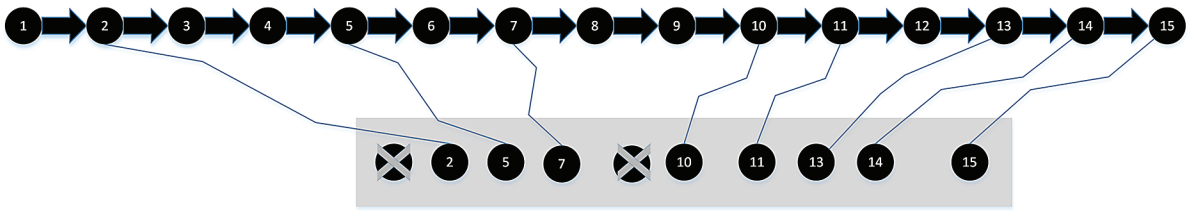


Figure 2. Reservoir sampling: data is selected from stream and replaced over time maintaining a sample representative of the stream history. A bias can be set to replace older points with more recent points to keep the sample more current.



DATA STREAM MINING TECHNIQUES

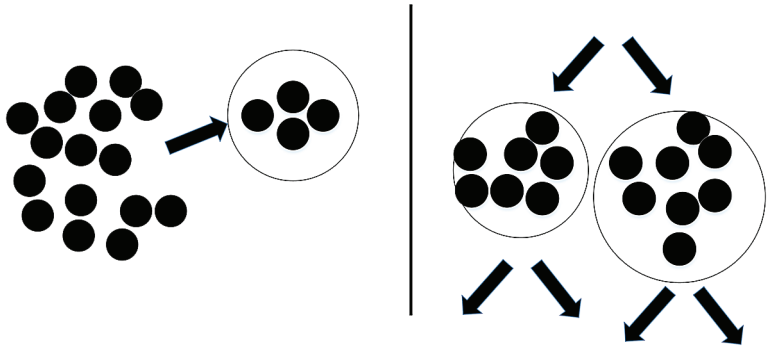
Once a stream is known to be of a manageable size it can be further processed in order to extract information. Data Mining can be categorised into two different groups known as Predictive and Descriptive techniques. Descriptive techniques aim to describe and classify the data by finding similarities between groups of data points. Predictive techniques use patterns in the data to predict the class labels or values. Following are examples of these algorithms found in Data Stream Mining from both categories.

Predictive Techniques

A common predictive method is to form a decision tree to classify data points as they arrive. In batch data there exist algorithms such as C4.5 and ID3 (Quinlan, 1993) which use metrics such as frequency, entropy or the GINI index to determine which attribute to test at each node of the tree (Bramer, 2013). From the root node down, the data set is split until each base node contains members of only one class. To increase the speed of classification and to make the tree more general a phase of pruning can optionally be included to remove the more specific base nodes. To make decision trees viable to Data Streams the Hoeffding Tree was developed in the system *Very Fast Decision Trees* (VFDT) (Domingos & Hulten, 2000). These trees incorporate the Hoeffding bound, mentioned above, and are able to produce a Decision Tree similar in structure to one produced in a batch method from a stream.

There have recently been a number of algorithms developed based on the PRISM algorithm (Cendrowska, 1987), a rule based algorithm for classification. This has been modified and improved upon since its conception into different forms, notably eRules (Stahl, Gaber, & Salvador, 2012) and the very recent G-eRules (Le, Stahl, Gomes, Gaber, & Di Fatta, 2008). PRISM was developed in response to a problem with decision trees known as the subclass tree problem. The way trees divide the data into two groups based on one attribute can be problematic as it may replicate the same decision on different branches. Rather than 'Divide and Conquer', as per decision trees, PRISM uses a 'Separate and Conquer' approach, where rules are developed to fit a portion of the data points resulting in these points being removed from any further processing (Figure 3).

Figure 3. Separate and conquer (left) and divide and conquer (right): divide and conquer results in two or more groups which may require duplicated rules to divide further. PRISM produces rules to cover a subset of points and removes those points from the main data set.

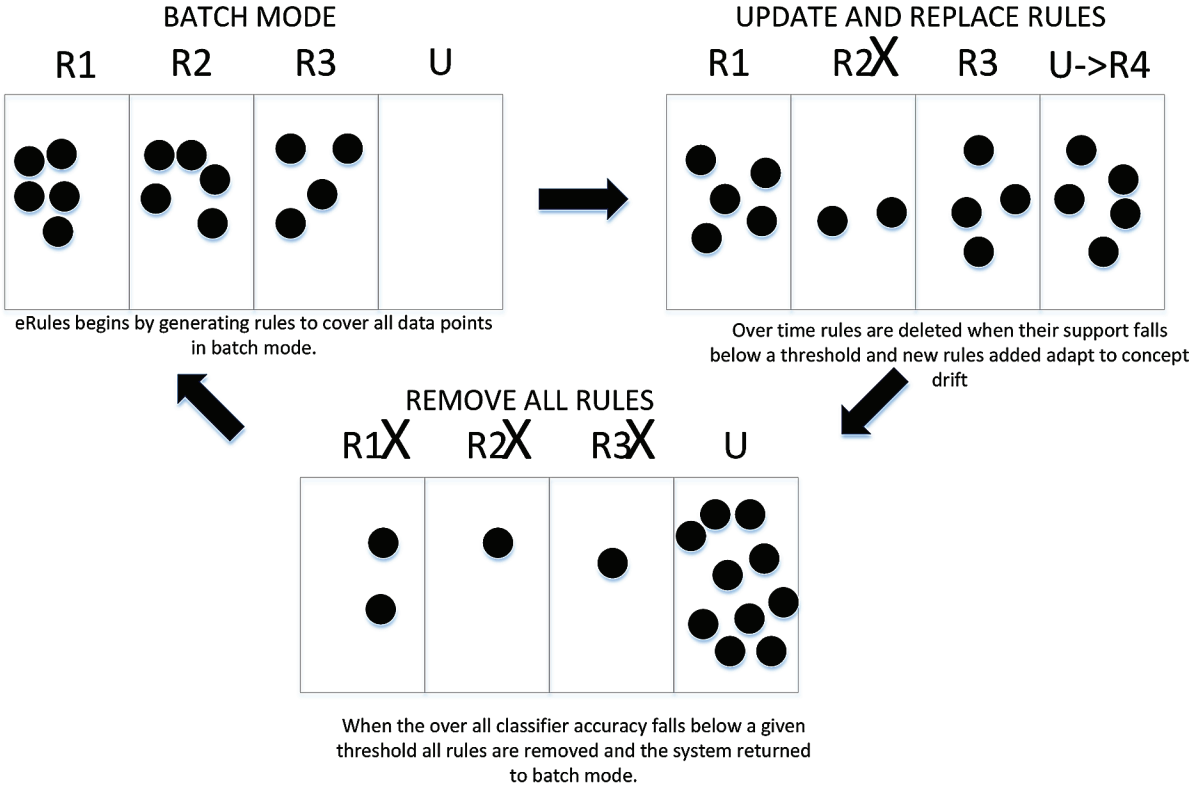


eRules is an adaption of PRISM for data streams and directly tackles the issues of concept drift, where the stream fundamentally changes in composition over time. The algorithm consists of three phases: the first learns a classification problem in batch mode; the second phase is triggered when the number of data points not covered by the established rules reaches a threshold, these points are then used to create additional rules; the third reviews all current rules and prunes those rules with a low classification accuracy. If the classification accuracy is altogether too low then the classifier is retrained in batch mode as per the first phase (Figure 4).

Descriptive Techniques

One of the most widely used descriptive techniques for static data are clustering algorithms. CluStream (Aggarwal, Han, Wang, & Yu, 2003), see Figure 5, is a form of k-means that has been adapted for streams through the use of micro-clusters. Data points are first grouped into many micro-clusters before being further grouped into *k* clusters. Both types of cluster are represented by cluster feature vectors holding statistical information on the cluster which allows the stream data to be reduced to a manageable size. Micro-clusters are a temporal extension of the cluster feature vector. They store key statistics of a cluster at a particular point in time, namely the sum and sum of squares for each attribute as well as

Figure 4. eRules: the three phases: to produce rules in batch mode to cover all data points, to remove and add rules as the number of unclassified data points increases and to return to batch mode if the total accuracy falls below a threshold

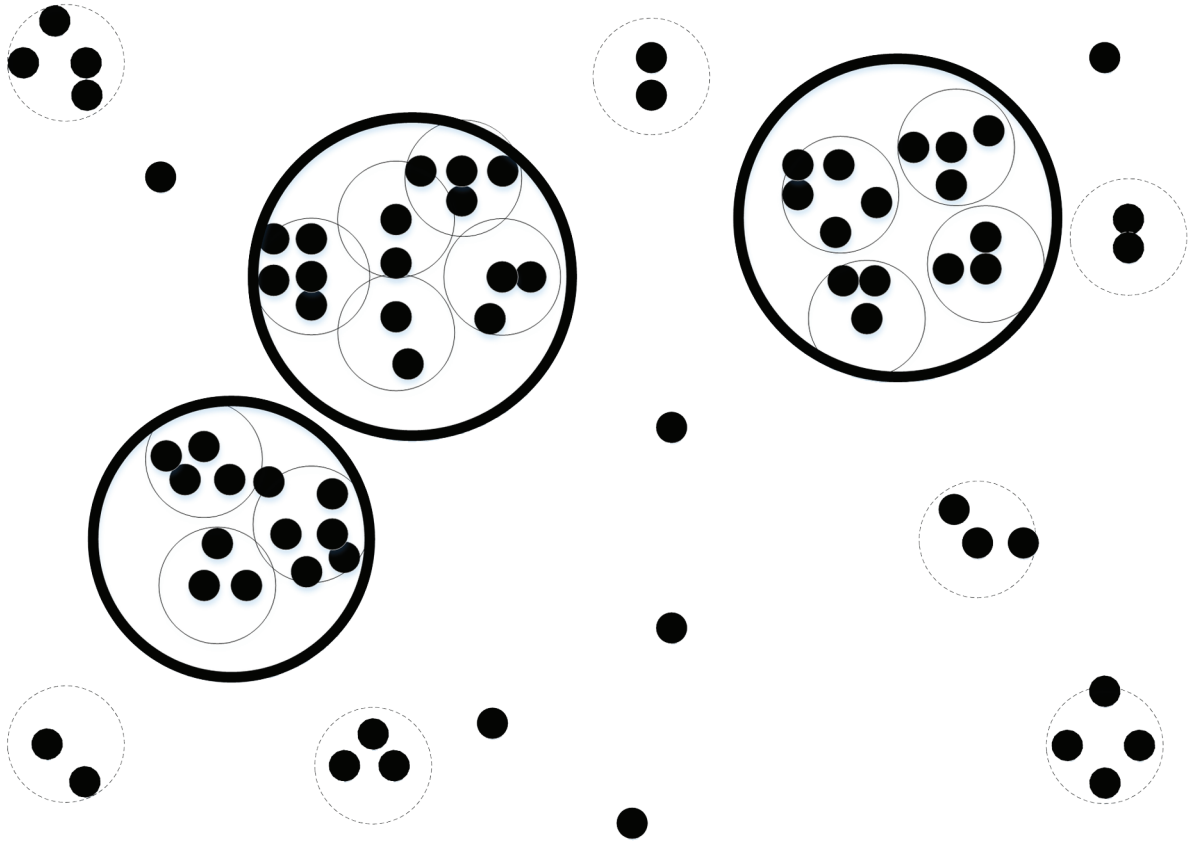


the sum and sum of squares of the current timestamp. A number of snapshots are stored along with the current micro-cluster (Aggarwal, 2014). If it is necessary to view a micro-cluster from a past snapshot it is trivial to work back using the vectors to the timestamp required. k-means is run on an initial batch of points to produce the initial micro-clusters. New points are added to the clusters as they arrive if they are within a set threshold distance; otherwise a new cluster is formed. Memory constraints dictate the maximum number of micro-cluster available, if the maximum is reached then a new cluster is formed at the expense of merging two clusters into one. Similar to CluStream, DenStream (Cao, Ester, Qian, & Zhou, 2006) is a stream clustering algorithm based DBSCAN (Ester, Kriegel, Sander, & Xiaowei, 1996). Micro-clusters have also been adapted for classification on Data Streams as in the MC-NN algorithm (Tennant, Stahl, & Gomes, 2015).

EVENTS

Events have been defined in a multitude of different ways and there seems to be no consensus as to which definition is wholly correct. Definitions include the overly broad “Anything that happens” or “a significant change in the state of the universe”(Hinze et al., 2009). A more detailed description can be

Figure 5. Groups of data points (black dots) are absorbed into micro-clusters (dotted lines) which are then in turn used to create K many global clusters (3 in this example).



found in (Etzion & Niblett, 2010): "An Event is an occurrence within a particular system or domain; it is something that has happened, or is considered as having happened in that domain." Wang, Liu, Liu, & Bai (2006) proposes "an occurrence of interest in time, which could be either a primitive event or a complex event." These definitions vary in levels of specificity but are often a question of semantics. The reader has likely a valid notion of what an event is before reading any of the above definitions and no attempt is made here to add to any of the definitions.

Event Streams are a subset of Data Streams where attributes are both quantative and qualitative. Event Mining is a very similar discipline to Data Stream Mining and many of the same considerations have to be taken into consideration, i.e. the four V's still apply to event streams. There are no guarantees as to how quickly events will be generated, and events by nature come in a large variety of types and instances. The Event Stream must also be assumed, for storage and processing purposes, to be infinite. An Event Stream is a Data Stream that focuses on discrete and/or quantative forms of data often in a more complicated data structure. Events are simply encapsulated packets of data arranged in tuples, often containing one or more timestamps and descriptions of the event. These tuples can be in XML, JSON (below) or various other formats.

```
<13.02.201518:12, TEMPERATURE_ALERT, SENSOR1, 75c>
```

Where a Data Stream describes the state of a particular entity, an event usually describes a particular action. For example a Data Stream may indicate the temperature of a component is gradually increasing, an event may be an alert triggered by an unusual temperature. By this example it is easy to see that a Data Stream may be converted to or may generate events and at least some Event Streams can be converted back into a Data Stream with a dimensionality equal to the number of values in the tuple. There are established techniques of making this conversion for example the simplistic method of (Paton & Díaz, 1999):

```
IF temp > X: CREATE_ALERT_EVENT() ;
```

The above example might be the input to a fire alarm system that triggers an alert when the temperature goes above a given threshold and effectively converts the data stream points to an event.

Early on in the evolution of Event Processing was the Publish/Subscribe Architecture. Devices were either Event Producers (RFID tags or website notifications) or event consumers (a phone app or a fire alarm console). Either the producer or the consumer can be an active part in the process by pushing or pulling events respectively. In most incarnations today it is usually a combination through middleware such as an Event Service (Etzion & Niblett, 2010; Eugster, Felber, Guerraoui, & Kermarrec, 2003) which may also filter events according to the subscribers specified preference. This concept has been built upon to bring about the two technologies discussed here.

Event Processing Tools

Big Data Analytics has traditionally been done on large databases or data warehouses as these have been one of the most efficient methods to retrieve and store data for long periods on a large scale. These systems have been adequate for the majority of Big Data for many years however they are not designed to tackle the additional challenges posed by Data Streams. The traditional relational database has undergone several revisions with several types now in existence, one of the latest being the NoSQL movement

that is very popular at the time of writing (Leavitt, 2010). Two types of databases have been developed to deal with an event rich or streaming environment. These are the *Active Database*, a more dynamic version of the traditional database, and the *Data Stream Management System* (DSMS) designed to work purely with streams. The latter introduces concepts used by CEP.

The Active Database system distinguishes itself from the classic Passive Database system by incorporating additional features into the standard application i.e. actions that are possible in a classical SQL application (specifically triggers and constraints) and that require additional overhead and explicit invocation are brought inside the database model and performed as default. Standard queries such as Update and Insert are invoked only at user request (Widom & Ceri, 1996). Triggers are a powerful tool in classical databases. They allow the user to set rules that will fire given a certain event, mostly an update of a row or value. Constraints are additional rules or parameters that can be incorporated into a field to enforce a realistic value, i.e. a rule may prevent a field representing temperature going below absolute zero. Both of these functions are available to classical databases but are added extras on top of the main application. In the active database they are a fundamental part of the database. This is advantageous when dealing with Event Streams and Event Processing as the overhead of each polling of the database is much reduced when compared to passive databases (Widom & Ceri, 1996). When the polling is as much as several times per second (or faster in many data streams) this can significantly impact the database and results in an intolerable amount of time spent locking the database and delaying further transactions (Dittrich, Gatziu, & Geppert, 1995).

Active Databases use the *Event – Condition – Action* (ECA) format of rules (Chakravarthy, Krishnaprasad, Anwar, & Kim, 1994). These are very simplistic and intelligible to produce. The event is the subject of change or simply a thing that changes (see the difficulty in defining this term as detailed earlier). The event can be any of the basic SQL commands or in addition something external to the database or a given unit of time elapsing. The Condition is the boundary or threshold that an attribute must pass to trigger the Action. This can be seen in the fire alarm example above. When the *temperature* surpasses *X Celsius* then an *alarm is triggered.* Some active databases forgo the full ECA rules and omit one of the first two components. Removing the event creates a production rule where the condition is checked at every possible event, removing the condition creates an Event Action Rule where the rule is triggered in response to a specified event regardless of whether or not a condition is met (Etzion, 1995; Paton & Díaz, 1999).

Data Stream Management Systems also utilize a recurring polling mechanism to process a data stream. Streams flow into the management system where they are buffered and queries run either on each stream or many streams using joint functionality. Rather than employing ECA rules the DSMS uses queries written in a bespoke language such as Continuous Query Language (CQL) (Arasu et al., 2004) or *Cayuga* (Demers et al., 2007). The focus of DSMS is on not storing the data from the stream, but rather on saving only the metadata produced by the queries. They are also more suited to dealing with unbounded streams. Unlike the Active Database the history of the stream is not stored (Cugola & Margara, 2012; Golab & Özsu, 2010).

This area of research is populated with numerous different kinds of languages, each with their own distinct implementations. Already mentioned are the widely used SQL in passive databases and the adapted SQL for streams such as CQL (Arasu et al., 2006). Another example is examined here – Event Processing Language (EPL) (Fülöp et al., 2010; Luckham & Schulte, 2011). DSMS systems often come with their own bespoke EPL though the majority incorporate the same features as outlined in (Owens, 2007). These are: the ability to retrieve event data, the ability to specify a time criteria in the query and

the ability to extract patterns from the events. The latter feature is used heavily when looking at CEP and further languages are specialised towards this aim. Surveys of these languages are available from both (Eckert, Bry, Brodt, Poppe, & Hausmann, 2011) and (Owens, 2007).

Complex Event Processing entails the amalgamation of events into more abstract and meaningful events (Schultz-Moller, 2008). The process consists of many smaller events that are recognisable when they are combined (Owens, 2007). There is a further comparison to be made in linguistics where each event represents a phoneme of language, when these individual utterances are combined they create meaning in the form of words and sentences. Earlier we mentioned the use of RFID tags in preventing shop lifting, this could be represented as a complex event and go on to be processed further, for instance the stream could be fed centrally for purposes of crime statistics.

CEP is not specifically targeted at Event Streams, its origins lie in methods of processing events across large business systems with a great many heterogeneous and parallel event producers. This setting is labelled an event cloud and may contain a great many event streams. The cloud is often so diverse that ordering of events is not so lightly assumed as in Event Streams, see Figure 6, instead the Cloud is defined as a *Partially Ordered SET* of events (POSET). Event Streams can be viewed as a specialised and simplified form of event clouds (Luckham, 2006).

The CEP algorithms are applied in a central CEP module. The input will be a series of events (primitive, complex etc.) and the output a combination of primitive and complex events ideally as a single ordered event stream. These events are then fed either forward for further processing or potentially back into the CEP module where they can be further aggregated. In this way CEP is as much a form of pre-processing as it is pattern detection. The higher level events go on to be used by other parts of the system as in Figure 7.

Already mentioned are two different kinds of events, the simple event and the complex event. Further distinctions between events are available in (Luckham & Schulte, 2011). Fülöp et al. (2012) define different types of *Event Processing Agents* (EPA) along with their corresponding events adding Mediated Event Processing in which events are enriched, transformed, and validated and Simple Event Processing

Figure 6. An event stream (left) and event cloud (right): the cloud has no guarantee of ordering due to the varying latencies on different streams.

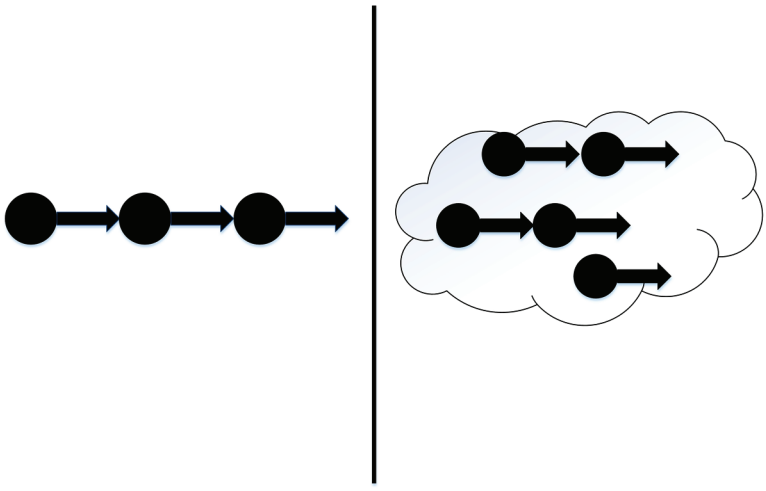
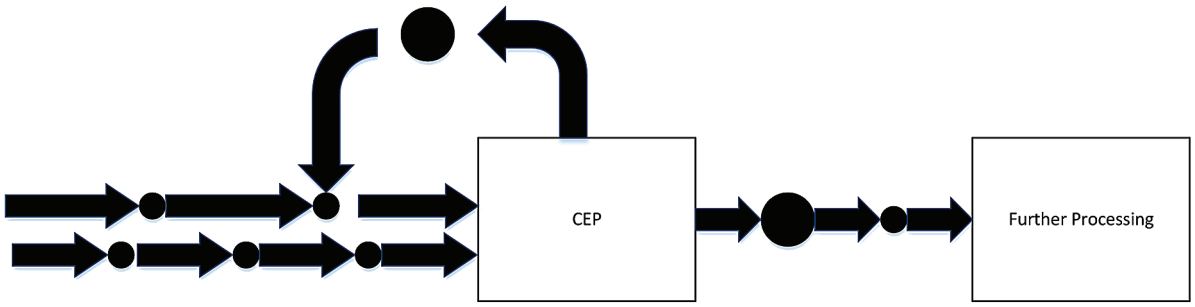


Figure 7. Complex event processing: one or more streams of heterogeneous events can be aggregated and/or sorted. Further processing, in the form of event stream processing, can take place on the output of the CEP engine.



in which events are filtered and routed. The three types (Simple, Mediated and Complex) can be combined together to form an event processing network – equating to three fundamental data mining steps: sourcing data, pre-processing and mining.

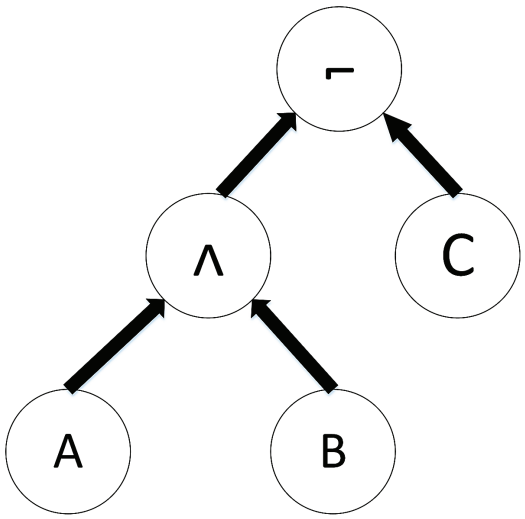
Simple or Primitive events are summed together often using event algebra or Event Language. Like the Event Processing Languages above there have been many new types of these languages proposed across the literature that focus on specific use cases or more general functionality. Repeated queries are run on the event stream with operators such as AND, OR and NOT to construct complex events. More advanced temporal languages have been created with further operators dealing with the streams temporal characteristics, operators such as *Sequences* (event *A* occurs after *B*) and *Time Sequence* (event *A* occurs within time *t* of event *B*). These vary from the ECA languages discussed above in that they operate at an event instance level, distinguishing between instances of events and event types (event types here meaning database transactions). The event component of an ECA rule can be defined using Event Processing Language granting the ECA rule greater fidelity. ECA rules are also markedly more human interpretable. Fülöp et al. (2010) provide a survey of these languages. Detection using EPL is commonly depicted graphically in a tree structure as in Figure 8.

```
(A AND B) occur WITHOUT C.
```

A Complex Event is constructed here if *A AND B occur WITHOUT C* within a given time interval. This complex event can then be used as an argument for further construction if placed on the node of another tree.

To mitigate potential errors from the weak ordering within a POSET one of two approaches may be used - *aggressive* or *conservative.* In an aggressive approach, errors are tolerable and the output must be maximised so little ordering mitigation takes place. An example of this would be any system where a quick response is necessary, such as in the use of an insulin pump where delays can be fatal (Wei et al., 2009). In a conservative approach it is critical to keep errors to a minimum and so ordering must be conserved at the cost of system throughput. This usually involves events being buffered before release. An example of this would be the RFID anti-shoplifting system. Clearly there is a trade-off to be made, the RFID system does not want to trigger false alarms, nor can it wait until the theft has successfully taken place to react. Systems have been developed that offer a *Quality of Service* (QoS) feature by offering a sliding scale of order guarantee (Liu, Li, Golovnya, Rundensteiner, & Claypool, 2009; Wei et al., 2009).

Figure 8. Complex event composition tree: read from bottom up



K-Slack is a conservative method of ensuring ordering (Li, Liu, Ding, Rundensteiner, & Mani, 2007). It buffers each event for *k* units of time using the timestamp values to determine when to release an event. The value of *k* can be set to match a meaningful value such as the maximum latency of a network, however, it is still a static value and likely not optimal for the duration of the stream. Punctuation is a second conservative method whereby events are buffered until a punctuation packet declares that no more of a certain timestamp will be seen on that particular stream. Following this, the events in the buffer can be released (Babu, Srivastava, & Widom, 2004; Wang, Zhou, & Nie, 2013).

BIG DATA ANALYTICS ON EVENT STREAMS

There has been some work done on developing data mining methods of extracting information from event streams, less so from CEP though this is possibly because CEP can be looked at as preparing streams for data mining to take place. As mentioned earlier CEP suffers from a confusion of terms throughout the literature and this may be the case in some of the following examples. The following are listed under the categories of ESP or CEP depending on the definitions given earlier.

Event Stream Processing

Twitter is a short broadcast micro-blogging application with an estimated 232 million *active* users (as recorded at the end of 2013; Edwards, 2013) producing in the region of 50 million discrete events, called *Tweets*, every day (Mathioudakis & Koudas, 2010). These are essentially strings (limited to 140 characters) with an attached label (*hashtag*) to declare the messages subject. Hashtags form hyperlinks and allow users to browse other connected tweets. These Tweet Events form a dense Event Stream made available through the Twitter API, *Firehose* (Twitter, 2015).

Adedoyin-Olowe et al.(2013), Bifet & Frank (2010) and Cameron et al. (2012) treat these broadcasts as text strings and process them to extract events. Social Media is fast to react and publish explicit news

and is monitored closely by News publishers. It is also recognised that through proper analysis it may implicitly broadcast events before the community themselves are aware of it, for example monitoring search queries to predict epidemics (Ginsberg et al., 2009).

Each Tweet can be treated as an event however further processing needs to be done to determine the event type. The strict language of events (*valve = open or temperature > 50*) does not apply here, instead there is a textual description of an event that must first be formalised. This task is commonly fulfilled by text mining. Marcus et al. (2011) developed an API to monitor manually set events via Twitter using a string matching technique, i.e. the user specifies the string to filter the tweets by and the API returns a graph depicting tweets over time with peaks representing particularly poignant moments. This method is one of the simplest and is an extension to the early filtering of Publish/Subscribe architecture mentioned above. Bifet & Frank (2010) in their approach to Sentiment Analysis (determining whether a tweet was positively or negatively inclined to an event or object) applied three different Data Stream Mining methods. These were a Multinomial Naïve Bayes, a Stochastic Gradient Descent and a Hoeffding Tree.

Adedoyin-Olowe et al. (2013) use only the hashtags from tweets, treating each hashtag as an event for *Association Rule Mining* (ARM). The Apriori algorithm (Agrawal, Imieliński, & Swami, 1993) is applied on the data set to uncover Association rules between subsets of hashtags. Apriori was developed as a means to lower the search space of all possible combinations of events to those whose subset surpass a threshold value for support. The strength of each rule is measured using Support and Confidence, support is defined as the probability that a randomly selected item set will be in all the items in a rule whilst confidence is the conditional probability that all the items will appear given the presence of one of the items. Ranking the rules by these two values will produce a list of reliable rules such as:

#KNN => #Datamining (Adedoyin-Olowe et al., 2013)

The above suggests that the presence of the hashtag #KNN is likely to coincide with the hashtag #Datamining, a very real possibility but not an interesting discovery. The support and confidence of the rules are subject to change over time as Twitter users react and broadcast to the world around them. This appears in the stream as a concept drift and is addressed here using *Tweet Change Discovery* (TCD) and *Transaction Based Rule Change Mining* (TRCM). In TCD the different rule sets generated from each Tweet are compared in a rule matching process and the degree of change is evaluated. This is used to place each rule into one of five categories: *Unexpected Consequent Change in Tweet, Unexpected Conditional Change in Tweet, Emerging Change in Tweet*, *New Rules,* and *'Dead' Rules.* TRCM uses this to monitor rule changes in real world events with a goal for these rules to be used as a decision support tool.

Complex Event Processing

The effectiveness of CEP to identify patterns and create meaningful complex events depends largely on the quality and relevance of the rules in place. These can be specified a priori or they can be formed with predictive analytics, i.e. mining the event stream or cloud.

Bayesian networks are an established method of machine learning where Bayes theorem is used to calculate probabilities based on the input. Each node is the likelihood of a state connected to the affecting probabilities above and influencing the probabilities below. The representative probabilities in the network are updated with each new data point it processes keeping the network current. A strength of a Bayesian Network is that it performs well over uncertain data. Naïve Bayesian networks are a specialised

form in which the network is simplified by the assumption that all input attributes are independent of each other (which in most cases is a large assumption). Wasserkrug, Gal, Etzion, & Turchin (2008) propose an algorithm to construct a Bayesian network from a set of events in order to predict subsequent events with some accuracy. This network is created anew with each set of events making it computationally inefficient however, it is noted that this can be improved by an approach which updates the existing network.

The authors of Debar, Becker, & Siboni (1992) use an *Artificial Neural Network* (ANN) in conjunction with two Expert Systems to identifying network incursions and issuing an alert to the system administrator. In this instance the data is stored in logs but is fed into the system as an Event Stream. The ANN is limited to numerical inputs and outputs and is unable to correctly interpret the network and raise alarms. This function is fulfilled by the accompanying Expert Systems that take as an input the ANN output as well as parts of the Network Stream unavailable to the ANN. The system has quite a few components to adapt around the ANN however it does perform well. Widder, von Ammon, Schaeffer, & Wolff (2008) similarly use an ANN in conjunction with Discriminant Analysis of Clusters to provide the ANN with an interpretable input. Discriminant Analysis determines the key variables that differentiate two groups or functions. A discriminant function is derived for each group and compared against the *Critical Discriminant Function* (CDF) to determine if this group is much altered from the norm. Widder, von Ammon, Schaeffer, & Wolff (2007) also use this technique to identify unusual (suspicious) transactions (those whose discriminant function is greater than the CDF) and record these patterns to be implemented as future rules.

When looking at a POSET of events (with aggressive or no ordering guarantee) the Event Cloud takes on quantifiable properties differing from a stream that can be exploited by statistical means. These include the altering size of the cloud (the total population of events within) and the change in breakdown of this population. The ordering, or near ordering, of events need not be a defining characteristic. For example, studying the dimensions of a histogram of event types can be used to trace the concept drift on an event cloud and derive patterns based on the makeup of the cloud at different times of the day. ARM can be applied to a POSET (Olmezogullari & Ari, 2013) as it can be to a data stream as in (R. Vilalta & Ma, 2002) in an effort to predict target events. Here it is assumed that the data is out of order and equates different combinations of the data set to each other. The rules developed from these patterns are then used to predict future target events, these events are specified a priori and patterns that do not precede an instance of the target event are filtered out.

A problem faced when dealing with Complex Events is the variety of events that may be available. Data Streams and Event Streams will contain specific primitive events or tuples of known size, a complex event can contain additional or modified characteristics, for example where a simple event has an occurrence time a complex event may have a duration which is problematic for the windowing techniques discussed earlier. Where clustering is readily applied to Data or Event Streams it is less readily applied to a cloud or stream of both primitive and complex events. A technique that can be adapted is MC-STREAM (Kwon, Lee, Balazinska, & Xu, 2008), an adaptation itself of CluStream. MC-Stream creates micro clusters of what are labelled contexts, a complex event made of groups of primitive events. The nature of these events make centroids very difficult to ascertain, where CluStream maintains a vector of statistics based off each cluster's centroids MC-STREAM uses the clusters medoid along with a distance metric based on the distances between each attribute, tuple and *aspect* (the collection of homogenous event types calculated using Earth Movers Distance; Rubner, Tomasi, & Guibas, 2000). Clusters are drawn up initially with events in batch mode and updated with each new context.

(Lee, You, Hong, & Jung, 2015) propose a clustering method to create complex events from primitives using k-means applied to a series of events leading to an *expert* (human) taking an action. The process involves a number of clustering steps to overcome the difficulty in comparing primitive and complex events mentioned above. Individual primitive events and actions (reactions to a complex event) are clustered and used as a basis for comparing event sequences and complex events. Using this similarity the event sequences are then clustered and refined using Markov Probabilities to identify and remove low impact primitive events. These new event sequences are used to create ECA rules for business use. Both methods of clustering complex events have to create a new characteristic with which to compare complex events to perform clustering and as such are more computationally expensive than the corresponding approaches on Data or Event Streams.

A final example of the application of machine learning to CEP is not to produce and refine events but rather speed up the application of event detection. (Schultz-Møller, Migliavacca, & Pietzuch, 2009) propose a method of improving the response time of individual queries by the application of operator specific algorithms, termed *Query Rewriting*. The cost of each operator is defined according the operators function but is always based upon CPU consumption. The Union operator is commutative and as such can be freely rearranged into its lowest cost form. This is found through an algorithm which arranges the queries into prefix trees in a manner similar to Huffman Coding(Huffman, 1952) . Union queries containing the same subsets of events are rearranged from bottom up with a pair of events with the combined lowest CPU cost forming the base much in the same way as a Huffman tree, the result is the optimal event pattern. The Next operator is optimised using genetic programming to find the lowest costing sub-pattern of the query. Optimising each query in this way is shown to significantly increase throughput and increase the total number of operators that can be applied to a stream.

DISCUSSION AND CONCLUSION

In this chapter we have given an outline of the three data stream processing techniques, Data Stream Mining, Event Stream Mining and Complex Event Processing. All three have to approach the issues of Velocity, Variety, Volume and Veracity though some of these prove larger problems than others. Veracity and Variety are more difficult to approach when dealing with CEP, the increased complexity of the data structures used increases Variety, making it harder to deal directly with an event's attributes whilst the expansiveness of the system and the Veracity this entails reduces a set of ordered event streams to an event cloud.

Veracity is addressed through a range of strategies that introduces some level of ordering guarantee however there is a large trade-off between performance and ordering. Aggressive ordering strategies make too much of an assumption of the cloud and will hinder any further processing. A basic range of statistical techniques are available for treating the POSET as an event cloud without any ordering concerns, however, the information available from this is limited. There is a lack of Data Mining Techniques that are targeted at event clouds, instead there are several adaptations of established Data Mining techniques for measurement streams and adaptations of ARM for streams. MC-STREAM is a promising approach that with further work may develop into a fully-fledged complex event clustering algorithm. The majority of other work has been undertaken with Bayesian networks and ANN, often with an accompanying system to interpret the results.

Active Databases and DSMS have been contrasted and a use case established for either one. They were developed to address the same problem but go about it in different ways. It is recognised that the majority of stream processing is best done with a DSMS though for smaller systems or systems where a history of the stream is essential an Active Database should be considered. CEP engines have been developed that can serve as a pre-processor to either of these.

There is very little work done on detecting and adapting to concept drift in CEP. Algorithms presented in Debar et al. (1992) and Widder et al. (2007) are designed to automate the production of rules from the data but these algorithms are not designed to adapt to a concept drift. These algorithms exist in ESP and Data Stream Mining (Adedoyin-Olowe et al., 2013) but none were found for CEP. The incorporation of Hoeffding bound in Data Stream algorithms (as in VFDT) has been accompanied by notable increases in performance, its inclusion into ESP and CEP is an area worth investigating.

Throughout the research for this chapter it has become very apparent that ESP and CEP are used interchangeably. This harkens back to the Tower of Babel issues mentioned in the introduction but also there is a misunderstanding as to what CEP entails. Whilst some refer to CEP as the processing of many loosely ordered events within a cloud other focus on the aggregation of primitive events into a complex one. It may well be both as described earlier, a kind pre-processing in preparation for further analyses that incorporates ordering of clouds and higher level events. The field is evolving alongside event processing and, as put by Luckham, (2006) may well become further synonymous with each other.

REFERENCES

Adedoyin-Olowe, M., Gaber, M. M., & Stahl, F. (2013). TRCM: A methodology for temporal analysis of evolving concepts in Twitter. In Artificial Intelligence and Soft Computing (LNAI), (Vol. 7895, pp. 135–145). Springer. doi:10.1007/978-3-642-38610-7_13

Adi, A., Botzer, D., Nechushtai, G., & Sharon, G. (2006). Complex Event Processing for Financial Services. In 2006 IEEE Services Computing Workshops (pp. 7–12). IEEE. doi:10.1109/SCW.2006.7

Aggarwal, C. C. (2014). A Survey of Stream Clustering Algorithms. In Data Clustering: Algorithms and Applications (pp. 231–255). CRC Press.

Aggarwal, C. C., Han, J., Wang, J., & Yu, P. S. (2003). A Framework for Clustering Evolving Data Streams. *Proceedings of the 29th International Conference on Very Large Data Bases*, *29*, 81–92. doi:10.1016/B978-012722442-8/50016-1

Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *SIGMOD Record*, *22*(2), 207–216. doi:10.1145/170036.170072

Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Mayur, D., Ito, K., … Widom, J. (2004). *STREAM: The Stanford Data Stream Management System*. Academic Press.

Arasu, A., Babu, S., & Widom, J. (2006). The CQL continuous query language: Semantic foundations and query execution. *The VLDB Journal*, *15*(2), 121–142. doi:10.1007/s00778-004-0147-z

Axelsson, S. (2000). Intrusion Detection Systems: A Survey and Taxonomy. *Computer Engineering*, 1–27.

Babcock, B., Babu, S., Datar, M., Motwani, R., & Widom, J. (2002). Models and Issues in Data Stream Systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (pp. 1–16). ACM. doi:10.1145/543613.543615

Babu, S., Srivastava, U., & Widom, J. (2004). Exploiting K-Constraints To Reduce Memory Overhead in Continuous Queries Over Data Streams. *ACM Transactions on Database Systems*, *29*(3), 545–580. doi:10.1145/1016028.1016032

Bifet, A., & Frank, E. (2010). *Sentiment Knowledge Discovery in Twitter Streaming Data. Discovery Science*. Berlin: Springer.

Bramer, M. A. (2013). *Principles of Data Mining* (2nd ed.). London: Springer. doi:10.1007/978-1-4471-4884-5

Bucklin, R. E., & Sismeiro, C. (2015). A Model of Web Site Browsing Behavior Estimated on Clickstream Data. *JMR, Journal of Marketing Research*, *40*(3), 249–267. doi:10.1509/jmkr.40.3.249.19241

Cameron, M. a., Power, R., Robinson, B., & Yin, J. (2012). Emergency situation awareness from twitter for crisis management.*Proceedings of the 21st International Conference Companion on World Wide Web - WWW '12 Companion*, 695. doi:10.1145/2187980.2188183

Cao, F., Ester, M., Qian, W., & Zhou, A. (2006). Density-based Clustering over an Evolving Data Stream with Noise. *SDM*, *6*, 328–339.

Cendrowska, J. (1987). PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, *27*(4), 349–370. doi:10.1016/S0020-7373(87)80003-2

Chakravarthy, S., Krishnaprasad, V., Anwar, E., & Kim, S.-K. (1994). Composite Events for Active Databases: Semantics, Contexts and Detection. In VLDB (pp. 606–617).

Chaudhry, N., Shaw, K., & Abdelguerfi, M. (2006). *Stream Data Management*. Springer.

Cugola, G., & Margara, A. (2012). Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Comput. Surv., 44*(1), 15:1–15:62.

Debar, H., Becker, M., & Siboni, D. (1992). A neural network component for an intrusion detection system. In *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy* (pp. 240–250). IEEE. doi:10.1109/RISP.1992.213257

Demers, A., Gehrke, J., Hong, M., Panda, B., Riedewald, M., Sharma, V., & White, W. (2007). Cayuga: A General Purpose Event Monitoring System. *CIDR 2007*, *Third Biennial Conference on Innovative Data Systems Research*.

Desouza, K. C., & Smith, K. L. (2014). *Big Data for Social Innovation | Stanford Social Innovation Review*. Retrieved February 9, 2015, from http://www.ssireview.org/articles/entry/big_data_for_social_innovation

Dittrich, K. R., Gatziu, S., & Geppert, A. (1995). The Active Database Management System Manifesto: A Rulebase of ADBMS Features.*2nd Workshop on Rules in Databases*. doi:10.1007/3-540-60365-4_116

Domingos, P., & Hulten, G. (2000). Mining high-speed data streams.*Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '00*. doi:10.1145/347090.347107

Ebbers, M., Abdel-Gayed, A., Budhi, V., & Dolot, F. (2013). *Addressing Data Volume, Velocity, and Variety with IBM InfoSphere Streams V3.0*. Academic Press.

Eckert, M., Bry, F., Brodt, S., Poppe, O., & Hausmann, S. (2011). A CEP babelfish: Languages for complex event processing and querying surveyed. *Studies in Computational Intelligence*, *347*(242438), 47–70. doi:10.1007/978-3-642-19724-6_3

Eckert, M., Oriented, S., Soa, A., & Eda, E. A. (2009). Complex Event Processing (CEP) Types of Complex Event Processing Event Queries. *Informatik-Spektrum*, *32*(2), 1–8. doi:10.1007/s00287-009-0329-6

Edwards, J. (2013). *Twitter's "Dark Pool": IPO doesn't mention 651 million users who abandoned twitter.* Retrieved April 20, 2015, from http://www.businessinsider.com/twitter-total-registered-users-v-monthly-active-users-2013-11

Elmagarmid, A. K. (2005). Stream Data Management. New York: Springer.

Ester, M., Kriegel, H.-P., Sander, J., & Xiaowei, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *KDD*, *96*(34), 226–231.

Etzion, O. (1995). *Reasoning about the behavior of active databases applications. Rules in Database Systems*. Springer.

Etzion, O., & Niblett, P. (2010). *Event Processing in Action. Online*. Stamford, CT: Manning Publications Co.

Eugster, P. T., Felber, P., Guerraoui, R., & Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM Computing Surveys*, *35*(2), 114–131. doi:10.1145/857076.857078

Fülöp, L. J., Beszédes, Á., Tóth, G., Demeter, H., Vidács, L., & Farkas, L. (2012). Predictive Complex Event Processing : A Conceptual Framework for Combining Complex Event Processing and Predictive Analytics.*Proceedings of the Fifth Balkan Conference in Informatics, September*, (pp. 26–31). doi:10.1145/2371316.2371323

Fülöp, L. J., Tóth, G., Rácz, R., Pánczél, J., Gergely, T., Beszédes, Á., & Farkas, L. (2010). Survey on Complex Event Processing and Predictive Analytics. In *Proceedings of the Fifth Balkan Conference in Informatics* (pp. 26–31). Citeseer.

Ginsberg, J., Mohebbi, M. H., Patel, R. S., Brammer, L., Smolinski, M. S., & Brilliant, L. (2009). Detecting influenza epidemics using search engine query data. *Nature*, *457*(7232), 1012–1014. doi:10.1038/nature07634 PMID:19020500

Golab, L., & Özsu, M. T. (2010). *Data Stream Management. Synthesis Lectures on Data Management* (Vol. 2). Morgan & Claypool Publishers.

Hinze, A., Sachs, K., & Buchmann, A. (2009). Event-based Applications and Enabling Technologies. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems* (pp. 1:1–1:15). New York: ACM Press. doi:10.1145/1619258.1619260

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, *58*(301), 13–30. doi:10.1080/01621459.1963.10500830

Huffman, D. A. (1952). A Method for the Construction of Minimum-Redundancy Codes.*Proc. IRE*, *40*, 1098–1101. doi:10.1109/JRPROC.1952.273898

Hutchison, D., & Mitchell, J. C. (2011). *Transactions on Large-Scale Data- and Knowledge-Centered Systems III* (A. Hameurlain, J. Küng, & R. Wagner, Eds.). Heidelberg, Germany: Springer-Verlag Berlin Heidelberg.

Hwang, J. H., Balazinska, M., Rasin, A., Çetintemel, U., Stonebraker, M., & Zdonik, S. (2005). High-availability algorithms for distributed stream processing. *Proceedings - International Conference on Data Engineering*, (pp. 779–790).

Johnson, T., Muthukrishnan, S., & Rozenbaum, I. (2005). Sampling Algorithms in a Stream Operator. *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, (pp. 1–12). doi:10.1145/1066157.1066159

Kwon, Y., Lee, W. Y., Balazinska, M., & Xu, G. (2008). Clustering events on streams using complex context information. In *Proceedings - IEEE International Conference on Data Mining Workshops, ICDM Workshops 2008* (pp. 238–247). Washington, DC: IEEE. doi:10.1109/ICDMW.2008.138

Le, T., Stahl, F., Gomes, J. B., Gaber, M. M., & Di Fatta, G. (2008). Computationally Efficient Rule-Based Classification for Continuous Streaming Data. In *Research and Development in Intelligent Systems XXIV* (p. 2014). Springer International Publishing.

Leavitt, N. (2010). Will NoSql live to Their Promise? *Computer*, *43*(2), 12–14. doi:10.1109/MC.2010.58

Lee, O., You, E., Hong, M., & Jung, J. J. (2015). Adaptive Complex Event Processing Based on Collaborative Rule Mining Engine. In Intelligent Information and Database Systems (Vol. 9011, pp. 430–439). Springer International Publishing. doi:10.1007/978-3-319-15702-3_42

Li, M. (2010). Robust Complex Event Pattern Detection over Streams. *Evaluation*, 176.

Li, M., Liu, M., Ding, L., Rundensteiner, E. A., & Mani, M. (2007). Event Stream Processing with Out-of-Order Data Arrival. In *27th International Conference on Distributed Computing Systems Workshops*. IEEE. doi:10.1109/ICDCSW.2007.35

Liu, M., Li, M., Golovnya, D., Rundensteiner, E. A., & Claypool, K. (2009). Sequence pattern query processing over out-of-order event streams. In *IEEE 25th International Conference on Data Engineering* (pp. 784–795). IEEE. doi:10.1109/ICDE.2009.95

Luckham, D. (2006). *What's the difference between ESP and CEP?* Retrieved April 20, 2015, from http://www.complexevents.com/2006/08/01/what's-the-difference-between-esp-and-cep/

Luckham, D., & Schulte, R. (2011). *Event Processing Technical Society - Event Processing Glossary - Version 2*. Retrieved April 15, 2015, from http://www.complexevents.com/wp-content/uploads/2011/08/EPTS_Event_Processing_Glossary_v2.pdf

Maletic, J. I., & Marcus, A. (2010). Data Cleansing Data Mining and Knowledge Discovery Handbook. In O. Mainmon & L. Rokach (Eds.), Data Mining and Knowledge Discovery Handbook (pp. 19–32). New York: Springer.

Mansouri-Samani, M., & Sloman, M. (1999). GEM: A generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, *4*(2), 96–108. doi:10.1088/0967-1846/4/2/004

Marcus, A., Bernstein, M. S., Badar, O., Karger, D. R., Madden, S., & Miller, R. C. (2011). Tweets as data: Demonstration of TweeQL and Twitinfo. In *Proceedings of the 2011 international conference on Management of data - SIGMOD '11* (p. 1259). New York: ACM. doi:10.1145/1989323.1989470

Mathioudakis, M., & Koudas, N. (2010). Twittermonitor: trend detection over the twitter stream.*Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, (pp. 1155–1158). doi:10.1145/1807167.1807306

Olmezogullari, E., & Ari, I. (2013). Online Association Rule Mining over Fast Data. In *IEEE International Congress on Big Data* (pp. 110–117). IEEE. doi:10.1109/BigData.Congress.2013.77

Owens, T. J. (2007). *Survey of Event Processing*. New York.

Paton, N. W., & Díaz, O. (1999). Active database systems. *ACM Computing Surveys*, *31*(1), 63–103. doi:10.1145/311531.311623

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Francisco: Morgan Kaufmann.

Rubner, Y., Tomasi, C., & Guibas, L. J. (2000). The Earth Mover's Distance as a Metric for Image Retrieval. *International Journal of Computer Vision*, *40*(2), 99–121. doi:10.1023/A:1026543900054

Schultz-Møller, N. P., Migliavacca, M., & Pietzuch, P. (2009). Distributed complex event processing with query rewriting.*Proceedings of the Third ACM International Conference on Distributed EventBased Systems DEBS 09*, 1. doi:10.1145/1619258.1619264

Schultz-Moller, P. N. (2008). *Distributed Detection of Event Patterns*. Imperial College of Science, Technology and Medicine.

Stahl, F., Gaber, M. M., & Salvador, M. M. (2012). eRules: A modular adaptive classification rule learning algorithm for data streams. *Res. and Dev. in Intelligent Syst. XXIX: Incorporating Applications and Innovations in Intel. Sys. XX - AI 2012*, *32nd SGAI Int. Conf. on Innovative Techniques and Applications of Artificial Intel*., (pp. 65–78).

Stonebraker, M., Çetintemel, U., & Zdonik, S. (2005). The 8 requirements of real-time stream processing. *SIGMOD Record*, *34*(4), 42–47. doi:10.1145/1107499.1107504

Tennant, M., Stahl, F., & Gomes, J. B. (2015). Fast Adaptive Real-Time Classification for Data Streams with Concept Drift. In *Internet and Distributed Computing Systems* (pp. 265–272). Springer. doi:10.1007/978-3-319-23237-9_23

Tsymbal, A., Pechenizkiy, M., Cunningham, P., & Puuronen, S. (2008). Dynamic integration of classifiers for handling concept drift. *Information Fusion*, *9*(1), 56–68. doi:10.1016/j.inffus.2006.11.002

Tucker, P., Maier, D., Sheard, T., & Fegaras, L. (2003). Exploiting punctuation semantics in continuous data streams. *IEEE Transactions on Knowledge and Data Engineering*, *15*(3), 555–568. doi:10.1109/TKDE.2003.1198390

Twitter. (2015). *The Twitter Firehose API*. Retrieved March 24, 2015, from https://dev.twitter.com/streaming/firehose

Vilalta, R., Ma, S., & Hellerstein, J. (2001). *Rule Induction of Computer Events*. Academic Press.

Vilalta, R., & Ma, S. M. S. (2002). Predicting rare events in temporal domains. In *IEEE International Conference on Data Mining* (pp. 474–481). IEEE.

Vitter, J. S. (1985). Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, *11*(1), 37–57. doi:10.1145/3147.3165

Wang, F., Liu, S., Liu, P., & Bai, Y. (2006). Bridging physical and virtual worlds: Complex event processing for RFID data streams. *Advances in Database Technology-EDBT*, *2006*, 588–607.

Wang, F., Zhou, C., & Nie, Y. (2013). Event Processing in Sensor Streams. In C. C. Aggarwal (Ed.), *Managing and Mining Sensor Data* (pp. 77–102). Springer Science & Business Media. doi:10.1007/978-1-4614-6309-2_4

Wasserkrug, S., Gal, A., Etzion, O., & Turchin, Y. (2008). Complex event processing over uncertain data. In *Proceedings of the second international conference on Distributed event-based systems* (Vol. 332, pp. 253–264). ACM. doi:10.1145/1385989.1386022

Wei, M., Liu, M., Li, M., Golovnya, D., Rundensteiner, E. A., & Claypool, K. T. (2009). Supporting a spectrum of out-of-order event processing technologies: from aggressive to conservative methodologies. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, (pp. 1031–1034). doi:10.1145/1559845.1559973

Widder, A., von Ammon, R., Schaeffer, P., & Wolff, C. (2007). Identification of suspicious, unknown event patterns in an event cloud. In *Proceedings of the inaugural international conference on Distributed eventbased systems* (pp. 164–170). Toronto: ACM. doi:10.1145/1266894.1266926

Widder, A., von Ammon, R., Schaeffer, P., & Wolff, C. (2008). Combining Discriminant Analysis and Neural Networks for Fraud Detection on the Base of Complex Event Processing. In *Proceedings of the 2nd international conference on Distributed event-based systems*. ACM.

Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, *23*(3), 69–101. doi:10.1007/BF00116900

Widom, J., & Ceri, S. (1996). *Active database systems: Triggers and rules for advanced database processing*. Morgan Kaufmann.

Yu, G., Li, C. W., Gu, Y., & Hong, B. (2011). Aggressive complex event processing with confidence over out-of-order streams. *Journal of Computer Science and Technology*, *26*(July), 685–696.

KEY TERMS AND DEFINITIONS

ANN: Artificial Neural Network.
ARM: Association Rule Mining.
CDF: Critical Discriminant Function.
CEP: Complex Event Processing.
DSMS: Data Stream Management Systems.
ECA: Event-Condition-Action rules.
EPA: Event Processing Agents.
ESP: Event Stream Processing.
POSET: Partially Ordered Set (of Events).
RFID: Radio Frequency Identification.
TCRM: Transaction Rule Change Mining.
VFDT: Very Fast Decision Trees.